Analyzing Performance of File Aggregation in VELOC

Mikaila Gossman, Jon C. Calhoun (advisor), Bogdan Nicolae (advisor), Melissa Smith (advisor)

CLEMSON UNIVERSITY

Argonne NATIONAL LABORATORY

The Need for Aggregation

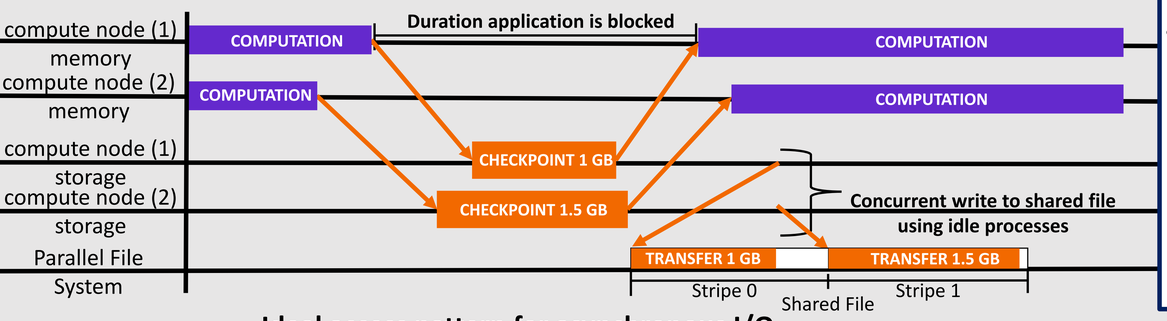
- Synchronous checkpointing methods (e.g. GenericIO) are projected to be infeasible at Exascale
- Asynchronous multi-level checkpoint runtimes like VELOC leverage fast local storage and flush to a parallel file system (PFS) independently in the background
- VELOC uses a one-file-per-process write strategy leading to I/O contention and many files on the PFS in applications that utilize large numbers of processes
- **We implement 2 working prototypes for asynchronous aggregation: (1) POSIX-based and (2)MPI-IO collective routine**
- **Study the behavior of said aggregation implementations on asynchronous checkpointing**
- **Compare results to VELOC's file-per-process asynchronous strategy, and GenericIO's synchronous aggregation strategy**

Approach

- Each process participating in checkpoint obtains offset in shared file using **Parallel Prefix-Sum**
- **POSIX-based**: baseline for asynchronous aggregation by **not** using any I/O optimizations

  Mmap local file to memory → open remote file lseek to offset → write
- **Ideal asynchronous aggregated workflow:** block application only for time it takes to checkpoint locally, eliminate false sharing with stripe-aligned writes
- **MPI-IO based**: direct comparison to GenericIO since it uses collective MPI-IO operations. MPI-IO will internally exchange data between processes to coordinate access to PFS

  mmap local file to memory → collective open remote file → collective write-at offset

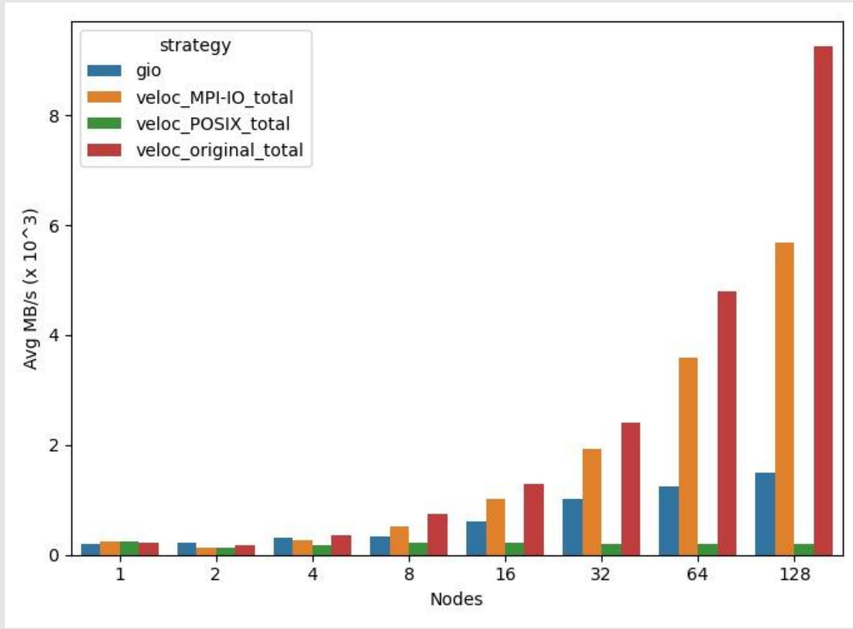


Ideal access pattern for asynchronous I/O

Experimental Setup

- All experiments are performed on Argonne© Theta, equipped with Intel Enterprise Edition Lustre PFS
- Microbenchmark spawns N (N = 1 → 128) processes, each checkpoints 1 GiB of memory
  - Top set of results number of nodes in varied
  - Bottom set of results number of processes per node is varied
- Stripe Size = 2 GB ; Storage Targets = min(N, 56), where 56 is the max available OSTs on Theta

Results

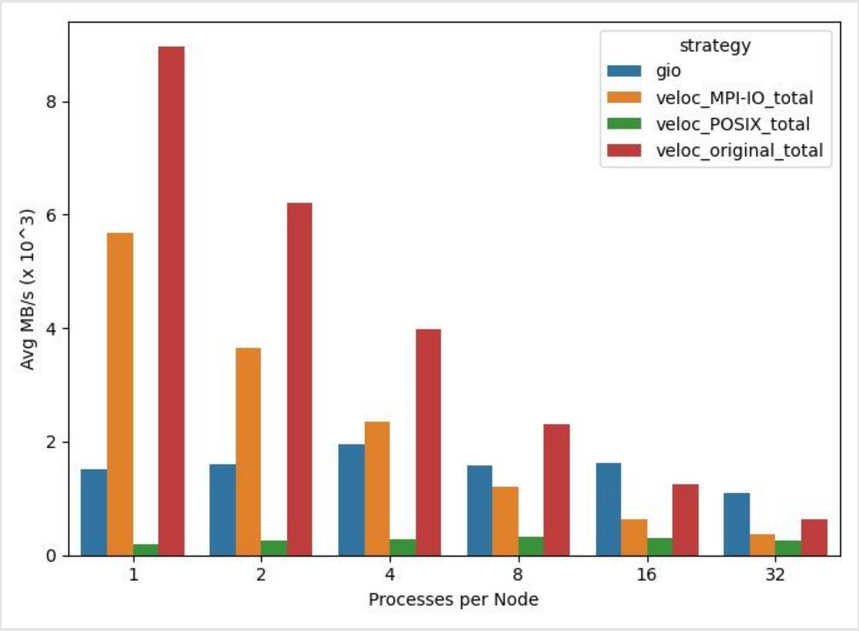


Bandwidth – 1 Process per Node:
With few numbers of processes per node less competition for local storage results in many flush requests at once :

- POSIX exhibits degraded performance since each process is accessing multiple OST's to copy their data at the offset
- MPI-IO maintains high throughput through data exchange and coordinated access

Bandwidth – Multiple Nodes & Processes
When more processes are added per each compute node:

- POSIX throughput improves slightly as processes on a node compete for local space, spreading out write requests to the PFS
- To avoid unnecessary synchronization due to the nature of collective MPI-IO calls the flushing mechanism on each compute node was altered to issue successive collective writes for each checkpoint file co-located on the node, thus serialization starts to overtake performance



Conclusions

- POSIX method suffers from congestion at the PFS level with few processes caused by false sharing and uncoordinated access patterns
- MPI-IO method suffers from multiple successive collective write calls for each node-local checkpoint, reducing chances for optimization
- In the future we will leverage the information from this preliminary study to design more sophisticated methods of aggregation that are optimized specifically for asynchronous, multi-level checkpointing

ACKNOWLEDGEMENTS:
Clemson University and Argonne National Lab are acknowledged for generous allotment of compute time on the Palmetto cluster and Theta supercomputer. This material is based upon work supported by the National Science Foundation under Grant No. SHF-1910197. The material was supported by U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11.357

FIND ME

CHECKOUT OUR CODE